# 1.13. Programação Orientada a Objetos em Python: Definindo Classes

Nós afirmamos anteriormente que o Python é uma linguagem de programação orientada a objetos. Até agora, usamos várias classes nativas para mostrar exemplos de estruturas de dados e controle. Um dos recursos mais poderosos em uma linguagem de programação orientada a objeto é a capacidade de permitir que um programador (solucionador de problemas) crie novas classes que modelem dados necessários para resolver o problema.

Lembre-se que usamos tipos de dados abstratos como descrição lógica de como um objeto de dados se parece (seu estado) e o que ele pode fazer (seus métodos). Ao construir uma classe que implementa um tipo de dado abstrato, um programador pode tirar proveito do processo de abstração e ao mesmo tempo, fornecer os detalhes necessários para usar a abstração em um programa. Sempre que desejarmos implementar um tipo de dados abstrato, vamos fazer isso com uma nova classe.

## 1.13.1. Uma Classe Fraction

Um exemplo muito comum para mostrar os detalhes da implementação de uma classe definida pelo usuário é construir uma classe para implementar o tipo de dados abstrato Fraction (fração). Nós já vimos que o Python fornece um número de classes numéricas para nosso uso. Há momentos, no entanto, que seria mais apropriado poder criar objetos de dados que se “pareçam” com frações.

Uma fração como

3/5

3/5 consiste de duas partes. O valor em cima, conhecido como numerador, pode ser qualquer inteiro. O valor embaixo, chamado denominador, pode ser qualquer número inteiro maior que 0 (frações negativas têm um numerador negativo). Embora seja possível criar uma aproximação em ponto flutuante para qualquer fração, neste caso nós gostaríamos de representar uma fração como um valor exato.

As operações para o tipo Fraction permitirão que um objeto de dado Fraction se comporte como qualquer outro valor numérico. Precisamos ser capazes de adicionar, subtrair, multiplicar e dividir frações. Nós também queremos ser capazes de mostrar frações usando a forma padrão com “barra”, por exemplo, 3/5. Além disso, todos os métodos de Fraction devem retornar resultados em seus menores termos de modo que, não importa qual computação seja realizada, sempre acabamos com a forma mais comum.

Em Python, definimos uma nova classe fornecendo um nome e um conjunto de métodos que são sintaticamente semelhantes às definições de função. Para este exemplo,

**class** **Fraction**:

*# Coloque os métodos aqui*

fornece a estrutura para definirmos os métodos. O primeiro método que todas as classes devem fornecer é o construtor. O construtor define a maneira como os objetos de dados são criados. Para criar um objeto Fraction, precisaremos fornecer dois dados, o numerador e denominador. Em Python, o método construtor é sempre chamado \_\_init\_\_ (com dois *underscores* antes e depois de init) e é mostrado na [Listagem 2](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#lst-pyconstructor).

**Listagem 2**

**class** **Fraction**:

**def** \_\_init\_\_(self, cima, baixo):

self.num = cima

self.den = baixo

Note que a lista formal de parâmetros contém três itens (self, cima, baixo). O self é um parâmetro especial que sempre deve ser usado como uma referência ao próprio objeto. Deve ser sempre o primeiro parâmetro formal; no entanto, esse parâmetro nunca receberá um valor na chamada. Como descrito anteriormente, as frações requerem dois objetos de dados de estado, o numerador e o denominador. A notação self.num no construtor define que um objeto Fraction tenha um objeto de dados interno chamado num como parte de seu estado. Da mesma forma, self.den cria o denominador. Os valores dos dois parâmetros formais são inicialmente atribuídos ao estado, permitindo que o novo objeto Fraction receba o seu valor inicial.

Para criar uma instância da classe Fraction, devemos invocar o construtor. Isso acontece quando usamos o nome da classe e passamos valores necessários para iniciar o estado (note que nunca invocamos \_\_init\_\_ diretamente). Por exemplo,

myfraction = Fraction(3,5)

cria um objeto chamado myfraction representando a fração

3/5 (três quintos). A [Figura 5](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#fig-fraction1) mostra esse objeto como implementado até agora.

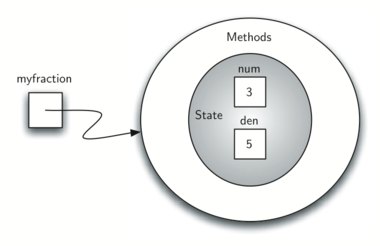


Figura 5: Uma Instância da Classe Fraction Referenciada por myfraction

A próxima coisa que precisamos fazer é implementar o comportamento requerido pelo tipo de dado abstrato. Para começar, considere o que acontece quando tentamos imprimir um objeto Fraction.

**>>>** myfraction = Fraction(3,5)

**>>>** print(myfraction)

<\_\_main\_\_.Fraction instance at 0x409b1acc>

O objeto Fraction, myfraction, não sabe como responder a esse pedido para imprimir. A função print requer que o objeto se converta em uma string (cadeia de caracteres) para que a string possa ser escrita na saída. A única escolha do myfraction é mostrar a referência real que é armazenada na variável (o próprio endereço). Isto não é o que nós queremos.

Existem duas maneiras de resolver este problema. Uma é definindo um método chamado show (mostrar) que permitirá que o objeto Fraction seja impresso como uma string. Podemos implementar este método como mostrado na [Listagem 3](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#lst-showmethod). Se criarmos um objeto Fraction como antes, nós podemos lhe pedir para se mostrar, ou em outras palavras, para imprimir seu valor no formato apropriado. Infelizmente, isso geralmente não funciona. Para que a impressão funcione corretamente, precisamos dizer à classe Fraction como se converter em uma string. Isto é o que a função print precisa para fazer o trabalho dela.

**Listagem 3**

**def** show(self):

**print**(self.num,"/",self.den)

**>>>** myfraction = Fraction(3,5)

**>>>** myfraction.show()

3 / 5

**>>>** print(myfraction)

<\_\_main\_\_.Fraction instance at 0x40bce9ac>

>>>

No Python, todas as classes têm um conjunto de métodos padrão que são fornecidos mas podem não funcionar corretamente. Um desses, \_\_str\_\_, é o método para converter um objeto em uma string. A implementação default para este método é retornar a string correspondente ao endereço da instância, como já vimos. O que precisamos fazer é fornecer uma implementação “melhor” para esse método. Dizemos que esta implementação **sobrescreve** a anterior, ou que redefine o comportamento do método.

Para fazer isso, nós simplesmente definimos um método com o nome \_\_str\_\_ e fornecemos uma nova implementação como mostrado na [Listagem 4](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#lst-str). Esta definição não precisa de nenhuma outra informação exceto o parâmetro especial self. Por sua vez, o método irá construir uma string convertendo cada pedaço dos dados de estado interno em strings e depois colocando um caractere / entre as strings por concatenação. A string resultante será retornada sempre que um objeto Fraction for solicitado para se converter em string. Observe que há várias maneiras de se usar essa função.

**Listagem 4**

**def** \_\_str\_\_(self):

**return** str(self.num)+"/"+str(self.den)

**>>>** myfraction = Fraction(3,5)

**>>>** print(myfraction)

3/5

**>>>** print("Eu comi", myfraction, "da pizza")

Eu comi 3/5 da pizza

**>>>** myfraction.\_\_str\_\_()

'3/5'

**>>>** str(myfraction)

'3/5'

>>>

Podemos sobrescrever muitos outros métodos para nossa nova classe Fraction. Algumas das mais importantes são as operações aritméticas básicas. Nós gostaríamos de poder criar dois objetos do tipo Fraction e depois adicioná-los usando a notação padrão “+”. Neste ponto, se tentarmos adicionar duas Frações, obtemos o seguinte:

**>>>** f1 = Fraction(1,4)

**>>>** f2 = Fraction(1,2)

**>>>** f1+f2

Traceback (most recent call last):

File "<pyshell#173>", line 1, in -toplevel-

f1+f2

TypeError: unsupported operand type(s) for +:

'instance' and 'instance'

>>>

A mensagem do interpretador Python *“unsupported operand type(s) for +”* (o tipo de operando não é suportado por +) informa que o problema é que o operador “+” não entende operandos Fraction.

Podemos consertar isso fornecendo à classe Fraction um método que sobrescreve o método de adição. Em Python, esse método é chamado \_\_add\_\_ e requer dois parâmetros. O primeiro, self, é sempre necessário, e o segundo representa o outro operando na expressão. Por exemplo,

f1.\_\_add\_\_(f2)

pediria ao objeto Fraction f1 para adicionar o objeto Fraction f2 ao seu próprio valor. Isso pode ser escrito na notação padrão, f1 + f2.

Duas frações devem ter o mesmo denominador para serem adicionadas. A maneira mais fácil de se certificar de que eles têm o mesmo denominador é simplesmente usar o produto dos dois denominadores como um denominador comum para que . A implementação é mostrada na [Listagem 5](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#lst-addmethod). A função de adição retorna um novo objeto Fraction com o numerador e o denominador da soma. Podemos usar esse método escrevendo uma expressão aritmética padrão envolvendo frações, atribuindo o resultado da adição e depois imprimindo o nosso resultado.

**Listagem 5**

**def** \_\_add\_\_(self,other):

novonum = self.num\*other.den + self.den\*other.num

novoden = self.den \* other.den

**return** Fraction(novonum,novoden)

**>>>** f1=Fraction(1,4)

**>>>** f2=Fraction(1,2)

**>>>** f3=f1+f2

**>>>** print(f3)

6/8

>>>

O método de adição funciona como desejamos, mas uma coisa poderia ser melhor. Note que

6/8 é o resultado correto (¼+½) mas que não está na forma irredutível (representação usando os “termos mais baixos”). A melhor representação seria 3/4. Para ter certeza de que nossos resultados estão sempre nos termos mais baixos, precisamos de uma função auxiliar que saiba como reduzir frações. Esta função precisará procurar o máximo divisor comum ou MDC. Podemos então dividir o numerador e o denominador pelo MDC e o resultado será reduzido para os termos mais baixos.

O algoritmo mais conhecido para encontrar o máximo divisor comum é o Algoritmo de Euclides, que será discutido em detalhes no Capítulo 8. O algoritmo de Euclides afirma que o máximo divisor comum de dois inteiros m e n é n se n é um divisor próprio de m. No entanto, se n não for um divisor próprio de m, então a resposta é o máximo divisor comum de n e o resto da divisão de m por n. Nós vamos simplesmente fornecer uma implementação iterativa aqui (veja [ActiveCode 1](https://panda.ime.usp.br/panda/static/pythonds_pt/01-Introducao/13-poo.html#lst-gcd)). Note que esta implementação do algoritmo de MDC funciona apenas quando o denominador é positivo. Isso é aceitável para nossa Classe Fraction porque dissemos que uma fração negativa será representada por um numerador negativo.

etc…

https://panda.ime.usp.br/panda/static/pythonds\_pt/01-Introducao/13-poo.html